Main ICG operations

```
ILProgram Program.lower();
```
- translate the whole program into an `ILProgram`

```
void ClassDecl.lower(ILProgram);
```
- translate method decls
- declare the class's method record (vtbl)

```
void MethodDecl.lower(ILProgram,
                      ClassSymbolTable);
```
- translate into IL fun decl, add to IL program

```
void Stmt.lower(ILFunDecl);
```
- translate into IL statement(s), add to IL fun decl

```
ILExpr Expr.evaluate(ILFunDecl);
```
- translate into IL expr, return it

```
ILType Type.lower();
ILType ResolvedType.lower();
```
- return corresponding IL type

An example ICG operation

```
class IntLiteralExpr extends Expr {
   int value;

   ILExpr lower(ILFunDecl fun) {
      return new ILIntConstantExpr(value);
   }
}
```


An example ICG operation

```
class AddExpr extends Expr {
   Expr arg1;
   Expr arg2;

   ILExpr lower(ILFunDecl fun) {
      ILExpr arg1_expr = arg1.lower(fun);
      ILExpr arg2_expr = arg2.lower(fun);
      return new ILIntAddExpr(arg1_expr, arg2_expr);
   }
}
```


An example overloaded ICG operation

```
class EqualExpr extends Expr {
   Expr arg1;
   Expr arg2;

   ILExpr lower(ILFunDecl fun) {
      ILExpr arg1_expr = arg1.lower(fun);
      ILExpr arg2_expr = arg2.lower(fun);
      if (arg1.getResultType().isIntType() &&
          arg2.getResultType().isIntType()) {
         return new ILIntEqualExpr(arg1_expr,
                                   arg2_expr);
      } else if (arg1.getResType().isBoolType() &&
                 arg2.getResType().isBoolType()) {
         return new ILIntEqualExpr(arg1_expr,
                                   arg2_expr);
      } else {
         throw new InternalCompilerError(...);
      }
   }
}
```

An example ICG operation

```
class VarDeclStmt extends Stmt {
   String name;
   Type type;

   void lower(ILFunDecl fun) {
      fun.declareLocal(type.lower(), name);
   }
}
```

`declareLocal` declares a new local variable in the IL function

ICG of variable references

```
class VarExpr extends Expr {
   String name;
   VarInterface var_iface;  // set during typechecking

   ILExpr lower(ILFunDecl fun) {
      return var_iface.generateRead(fun);
   }
}

class AssignStmt extends Stmt {
   String lhs;
   Expr rhs;
   VarInterface lhs_iface;  // set during typechecking

   void lower(ILFunDecl fun) {
      ILExpr rhs_expr = rhs.lower(fun);
      lhs_iface.generateAssignment(rhs_expr, fun);
   }
}
```

`generateRead`/`generateAssignment` generate IL code to read/assign the variable

- code depends on the kind of variable (local vs. instance)

ICG of local variable references

```
abstract class VarInterface {
   String name;
   abstract ILExpr generateRead(ILFunDecl fun);
   abstract void generateAssignment(ILExpr rhs,
                                    ILFunDecl fun);
}

class LocalVarInterface extends VarInterface {
   ILExpr generateRead(ILFunDecl fun) {
      ILVar var = fun.lookupVar(name);
      return new ILVarExpr(var);
   }
   void generateAssignment(ILExpr rhs_expr,
                           ILFunDecl fun) {
      ILVar var = fun.lookupVar(name);
      fun.addStmt(
         new ILAssignStmt(new ILVarExpr(var),
                          rhs_expr));
   }
}
```


ICG of instance variable references

```
class InstanceVarInterface extends VarInterface {
   ClassSymbolTable class_st;
   ILExpr generateRead(ILFunDecl fun) {
      ILExpr rcvr_expr =
        new ILVarExpr(fun.lookupVar("this"));
      ILType class_type =
         ILType.classILType(class_st);
      ILRecordMember var_member =
         class_type.getRecordMember(name);
      return new ILFieldAccessExpr(rcvr_expr,
                                   class_type,
                                   var_member);
   }
   void generateAssignment(ILExpr rhs_expr,
                           ILFunDecl fun) {
      ILExpr rcvr_expr =
        new ILVarExpr(fun.lookupVar("this"));
      ILType class_type =
         ILType.classILType(class_st);
      ILRecordMember var_member =
         class_type.getRecordMember(name);
      ILAssignableExpr lhs =
         new ILFieldAccessExpr(rcvr_expr,
                               class_type,
                               var_member);
      fun.addStmt(new ILAssignStmt(lhs, rhs_expr));
   }
}
```

ICG of if statements

What IL code to generate for an if statement?

```
if (testExpr) thenStmt else elseStmt
```


ICG of if statements

```
class IfStmt extends Stmt {
   Expr test;
   Stmt then_stmt;
   Stmt else_stmt;

   void lower(ILFunDecl fun) {
      ILExpr test_expr = test.lower(fun);
      ILLabel false_label = fun.newLabel();
      fun.addStmt(
         new ILCondBranchFalseStmt(test_expr,
                                   false_label));
      then_stmt.lower(fun);
      ILLabel done_label = fun.newLabel();
      fun.addStmt(new ILGotoStmt(done_label));
      fun.addStmt(new ILLabelStmt(false_label));
      else_stmt.lower(fun);
      fun.addStmt(new ILLabelStmt(done_label));
   }
}
```


ICG of print statements

What IL code to generate for a print statement?

```
System.out.println(expr);
```

No IL operations exist that do printing (or any kind of I/O)!

Runtime libraries

Can provide some functionality of compiled program in **external runtime libraries**

- libraries written in any language, compiled separately
- libraries can contain functions, data declarations

Compiled code includes calls to functions & references to data declared libraries

Final application links together compiled code and runtime libraries

Often can implement functionality either through compiled code or through calls to library functions

- tradeoffs?

ICG of print statements

```
class PrintlnStmt extends Stmt {
   Expr arg;

   void lower(ILFunDecl fun) {
      ILExpr arg_expr = arg.lower(fun);
      ILExpr call_expr =
         new ILRuntimeCallExpr("println_int",
                               arg_expr);
      fun.addStmt(new ILExprStmt(call_expr));
   }
}
```

What about printing doubles?

ICG of new expressions

What IL code to generate for a `new` expression?

```
class C extends B {
   inst var decls
   method decls
}
... new C() ...
```


ICG of new expressions

```
class NewExpr extends Expr {
   String class_name;

   ILExpr lower(ILFunDecl fun) {
      generate code to:
         allocate instance record
         initialize vtbl field with class's method record
         initialize inst vars to default values
      return reference to allocated record
   }
}
```


An example ICG operation

```
class MethodCallExpr extends Expr {
   String class_name;

   ILExpr lower(ILFunDecl fun) {
      generate code to:
         evaluate receiver and arg exprs
         test whether receiver is null
         load vtbl member of receiver
         load called method member of vtbl
         call fun ptr, passing receiver and args
      return call expr
   }
}
```

IGC of array operations

What IL code to generate for array operations?

`new` *`type`*`[`*`expr`*`]`

`arrayExpr``.length`

`arrayExpr``[`*`indexExpr`*`]`